fsim_logic – A VHDL type for testing of FLYTRAP

Joanne E. DeGroat, Ph.D.
Department of Electrical and Computer Engineering
The Ohio State University
Columbus, OH USA
degroat.1@osu.edu

Abstract— **When designing modern digital components and systems it is often desirable to know how the circuit performs when a logic fault occurs. There are numerous custom simulators that allow this exploration. Current digital components and systems are designed using modern HDLs. Being able to conduct fault simulation using the HDL of design, allows for this evaluation to be done without additional translation to the design input required by a custom simulator. VHDL allows for the creation of custom logic systems packages. fbit and fsim_logic_package are logic system packages that allow for fault injection at the gate level. The packages allow the threshold for fault injection to be adjusted from very high fault injection rates, 1 in 10 gate evaluations, to very low fault injection rates, 1 in 1,000,000,000 or more gate evaluation. Additionally, this logic type is essential when designing fault tolerant circuits where the design can only be verified through the random injection of faults into the circuit. It can also be very useful in evaluating the actions and performance of a system when a low level fault occurs.**

Keywords—fault simulation; digital fault, faut tolerant circuits.

I. Introduction

Fault tolerant circuit research was one focus of digital circuit design and digital VLSI in the 1980s. In the 1970s and 80s there was much interest in fault simulators, programs that would simulate circuits and no fault conditions and under fault conditions [1]. At that time, and prior to the era of HDLs, custom fault simulators were built. Even today custom simulators are being built to study the performance of a processor under fault. On such simulator is the SESC simulator [2].

Modern circuits are designed in HDLs and it would be nice if minimal additional effort was needed to inject faults into the architecture at the lowest level. fbit and fsim_logic are VHDL logic systems that provide a logic types that allows for random injection of faults at the gate level. This logic system was developed to test fault tolerant circuits being developed and was essential for verification of that work. The alternative is to create a large number of instances of the design, each with a different injected hard error. Each of these instances must then be simulated. Many possible locations of where the fault will occur are overlooked in this approach. fsim_logic provides the capability to have random injection of faults into the DUT and significantly reduces the workload of doing the fault simulation.

The package also has application to modern design. The focus of some work today is on redundant computation paths and lowering the operating voltage until faults just start to occur. What is the action of the system when a fault does occur? Occurrence of a fault requires the computation to be rolled back as done in many modern systems. This is occurring more frequently as we strive for minimal energy computation and reliable computation. [3,4]

II. Background

Early work on the use of VHDL for fault simulation concentrated on development of CAD tools that generated duplicates of the original HDL description with a given “gate” in error [5,6]. This HDL description would then be simulated to observe the behavior under fault. By nature this methodology modeled the circuit containing stuck-on-faults. If the circuit contained 100,000 logic nodes, 100,000 models would need to be created to fully evaluate the circuit and 100,000 simulations would need to be run. This approach was not viable previously and today’s systems are now at a point where this approach is beyond viable.

An alternative to creating the many thousands of instances of the HDL circuit description is to inject error into the evaluation and generation of the output from a gate during simulation. VHDL offers this capability though the use of packages and custom types. The type names chosen for the fault injection type is fbit and fsim_logic. This is much like type bit and type std_logic of VHLD, only in this package when the output of a logic operation is being evaluated, the output could be in error. Whether an error occurs of not is determined by generation of a random number. If the random number exceeds a threshold, an error is injected, i.e., the output of the gate will be in error, i.e., it will the complement of the expected value for fbit and when using fsim_logic, a table for determination of the value under error is used. For example if the value should be high impedance, Z, it could be a capacitive high, H, under error.

There is also interest in creating circuit models that are representative of the occurrence of errors in circuits. This requires the use of complex probability density functions to accurately model the occurrence of errors. [7,8] The probability density function used in the fbit and fsim_logic packages does not have that requirement. In this work the objective is to observe the circuit’s performance under fault and to do so in reasonable simulation time. Through use of a uniform distribution for generation of the random number and adjustment of the threshold, the injection rate of errors into the circuit can be controlled to enable evaluation of operation of the circuit under fault in a modest amount of simulation time. Results have shown this methodology to be effective.

III. The Fualt Simulation Types

A. Package fbit_logic

Package fbit_logic contains types fbit and fbit_vector. The package declarative design unit is shown in Figure 1. The package contains a declaration for type fbit and fbit_vector. This type would be used to test designs done using bit and bit_vector. By simply pre-pending the 'f' in front of the bit and bit_vector in the declarations, the modification is complete. As is seen in the package, the basic logic functions are overloaded for the new type so no further change is needed to the design. To be complete, the relational operators such as <, >, =, etc. also need to be overloaded. [9]

At the end of the package declaration is the declaration of the threshold and the random number generator function. The random number generator function could have been moved to the package body but was placed here for visibility reasons. The threshold CONSTANT is the value that needs to be changed to modify the number of errors injected into a design.

```
-------------------------------------------------------------------------------------
-- Fault Simulation Package – TYPE fbit (non-resolved)
-------------------------------------------------------------------------------------
PACKAGE fbit_logic IS

TYPE fbit IS ('0',  -- low
              '1'   -- high  );
-------------------------------------------------------------------------------------
--unconstrained array of fbit
-------------------------------------------------------------------------------------
TYPE fbit_vector IS ARRAY (NATURAL RANGE <>) of fbit;
-------------------------------------------------------------------------------------
--Declare logic functions
-------------------------------------------------------------------------------------
FUNCTION "AND" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "OR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NAND" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "XOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "XNOR" (l:fbit; r:fbit) RETURN fbit;
FUNCTION "NOT" (l:fbit) RETURN fbit;
-------------------------------------------------------------------------------------
FUNCTION "AND" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "OR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NAND" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "XOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "XNOR" (l,r:fbit_vector) RETURN fbit_vector;
FUNCTION "NOR" (l:fbit_vector) RETURN fbit_vector;
-------------------------------------------------------------------------------------
CONSTANT threshold : REAL := -.999999;
impure FUNCTION getrand RETURN real;

END fbit_logic;
```

Figure 1. Example of a figure caption. *(figure caption)*

In the body of the fsim_logic package are the routines that determine the output of a gate evaluation. In Figure 2 a portion of the fbit_logic package is shown. Space precludes including the complete package. All of the functions are evaluated using tables. After the logic function is evaluated and a result determined from the table, a random number is generated using the VHDL UNIFORM random number generation function. This random number is then compared to the threshold. If greater than the threshold then the output is complemented, again using table lookup.

```
LIBRARY ieee;
USE ieee.math_real.ALL;
PACKAGE BODY fbit_logic IS
-------------------------------------------------------------------------------------
--Local types
-------------------------------------------------------------------------------------
TYPE fbit_1d IS ARRAY (fbit) of fbit;
TYPE fbitlogic_table IS ARRAY (fbit,fbit) of fbit;
shared VARIABLE seed1 : INTEGER := 500045;
shared VARIABLE seed2 : INTEGER := 100001;
-------------------------------------------------------------------------------------
--TABLE for error return
-------------------------------------------------------------------------------------
CONSTANT error_table : fbit_1d := ('1','0');
-------------------------------------------------------------------------------------
--AND FUNCTION
-------------------------------------------------------------------------------------
CONSTANT and_table : fbitlogic_table := (
   --    ----------------------------------------
   --    |  0       1
   --    ----------------------------------------
         ('0',    '0'),
         ('0',    '1')      );

FUNCTION "AND" (l : fbit, r : fbit) RETURN fbit IS
  VARIABLE val : fibt;
  VARIABLE rnd : REAL;
BEGIN
  val := and_table (l,r);
  rnd := getrand;
  IF (rnd > threshold) THEN val := (error_teble(val)); END IF;
  RETURN (val);
END "AND";
***
-------------------------------------------------------------------------------------
-- Function to return random number
impure FUNCTION getrand RETURN real IS
  VARIABLE vrandval : REAL;
BEGIN
  UNIFORM (seed1,seed2,vrandval);
  RETURN (vrandval);
END getrand;
***
END fbit_logic;
```

Figure 2. Package fbit_logic package body design unit.

The fbit_logic package first needed to be verified for correctness and for the rate of error injection. Only one error which was found in the initial version of fbit_logic and that was in one of the relational functions where one of the results was in error.

B. *Package fsim_logic*

fsim_logic is a package that for the most part mirrors the standard package, std_logic. As with std_logic it provides for resolved types. Once again the standard operators are overloaded and the operations are specified in tables as illustrated in Figure 3.

```
--truth table for “and” function
CONSTANT and_table : fsimlogic_table := (
-- --------------------------------------------
--| U   X   0   1   Z   W   L   H   -
--  --------------------------------------------
  (‘U’,‘U’,‘0’,‘U’,‘U’,‘U’,‘0’,‘U’,‘U’),  -- U
  (‘U’,‘X’,‘0’,‘X’,‘X’,‘X’,‘0’,‘X’,‘X’),  -- X
  (‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’),  -- 0
  (‘U’,‘X’,‘0’,‘1’,‘X’,‘X’,‘0’,‘1’,‘X’),  -- 1
  (‘U’,‘X’,‘0’,‘X’,‘X’,‘X’,‘0’,‘X’,‘X’),  -- Z
  (‘U’,‘X’,‘0’,‘X’,‘X’,‘X’,‘0’,‘X’,‘X’),  -- W
  (‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’,‘0’),  -- L
  (‘U’,‘X’,‘0’,‘1’,‘X’,‘X’,‘0’,‘1’,‘X’),  -- H
  (‘U’,‘X’,‘0’,‘X’,‘X’,‘X’,‘0’,‘X’,‘X’)); -- -
```

Figure 3. Table for ‘AND’ logic operation

The “and_table” shown in Figure 3, and the other logic tables, mirror those in std_logic. The one significant difference in the package is the error_table which specified the injected error. In the package for type fbit the error table simple inverted the output of the gate. As there are only two values in the logic value system the choice is easy. For fsim_logic there are many choices for the value to be injected. One of the goals of this work was to have the ability to evaluate a circuit’s performance under fault conditions. Choosing a value such as ‘X’ would result in the permeation of the ‘X’ value throughout the circuit and the simulation results, although useful, would not necessarily reflect the most meaningful information. For this reason only the driven high and low and capacitive high and low are inverted under error as shown in Figure 4.

```
CONSTANT error_table : fsimlogic_1d := (
-- --------------------------------------------
--| U   X   0   1   Z   W   L   H   -
--  --------------------------------------------
  (‘U’,‘X’,‘1’,‘0’,‘Z’,‘W’,‘H’,‘L’,‘-’);
```

Figure 4. Error table for type fsim_logic

This package was also verified and two error were found. The first was that in the and_table, shown in Figure 3, the operation of ‘1’ AND ‘H’ should result in a ‘1’ as is the case for the corrected table here. In the original version of the package the result was an ‘X’ and erroneous. The other error was in the type conversion function. In the original version of the type conversion function “to_bit” which allows conversion from fsim_logic to bit, a ‘0’ resulted in a ‘1’ and a ‘1’ resulted in a ‘0’. These errors were corrected, allowing characterization of the fault injection rates.

C. *Characterization of the fault injection*

The threshold for error injection is set though the value set on the CONSTANT threshold. Setting the threshold to 1.0 results in no error ever being injected into the circuit. Setting a value of 0.9 would result in 1 of 10 gate evaluations resulting in an error. As part of the verification of the packages for fbit and fsim_logic, the error injection rate was also verified.

When the threshold is set to 1.0 no errors were ever injected as should be. Table 1 shows the result of testing for the AND and OR logic functions. As these reflect simulation starting with the same random number seed, they should produce the same results. (That is a side benefit in that the results of error injection simulations are repeatable.) As can be seen in the table when the threshold is set to 0.999 an error injection rate of ~0.1% should result. The results of 0.07% for 10,000 instances and 0.095% for 100,000 instances correspond nicely to expectations. When the threshold is set to 0.9 the injection rate should be approximately 1 in 10 gate evaluations. As expected errors are injected approximately 10% of the time.

TABLE I. ERROR INJECTION RATES

Threshold	Gates	Errors	Error Rate(%)
‘AND’ 0.999	100	0	0
	10000	7	0.07%
	100000	95	0.095%
‘AND’ .90	100	11	11%
	10000	998	9.98%
	100000	9924	9.924%
‘OR 0.999	100	0	0
	10000	7	0.07%
	100000	95	0.095%
OR .90	100	11	11%
	10000	998	9.98%
	100000	9924	9.924%

Verifying each function one at time has merit and is needed for a verification of the package, but a more complete verification is required. For this case a small circuit was simulated, a 1-bit full adder. Frist the threshold was set to 1 to verify that the adder was coded correctly. After this simulation the threshold was readjusted to 0.9999. A full adder has 6 logic gate evaluations that take place during execution. Exhaustive simulation generated 10 instances of evaluation. With the threshold set to this level not all run will result in any error being generated. As shown in Figure 5, the simulation results resulted in no error.

Resetting the threshold to a setting of 0.9099 resulted in error being injected into the simulation. As there are 6 gate evaluations that take place for each input vector most vectors will likely have an error. As can be seen when comparing

Figure 5 and Figure 6, many of the vector evaluations result in an error.

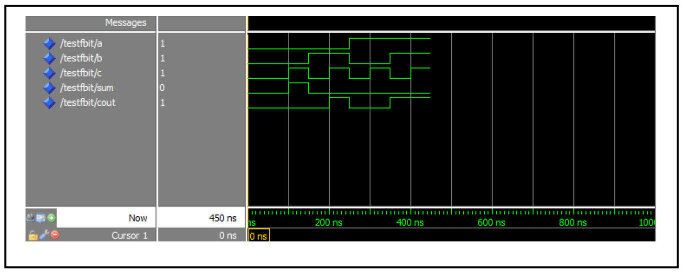

```
Messages
/testfbit/a      1
/testfbit/b      1
/testfbit/c      1
/testfbit/sum    0
/testfbit/cout   1
Now       450 ns
Cursor 1    0 ns
```

Figure 5. Table for 'AND' logic operation – threshold 0.9999

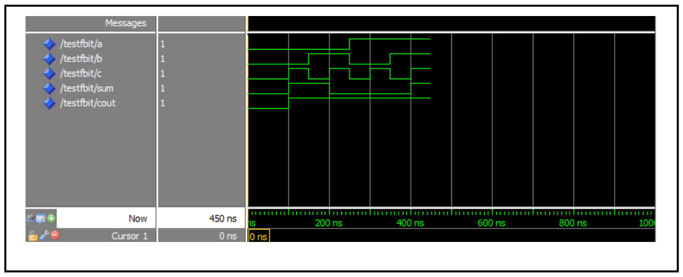

```
Messages
/testfbit/a      1
/testfbit/b      1
/testfbit/c      1
/testfbit/sum    1
/testfbit/cout   1
Now       450 ns
Cursor 1    0 ns
```

Figure 6. Table for 'AND' logic operation –threshold 0.9099

IV. APPLICATION OF FBIT AND FSIM_LOGIC

The packages were used to test and verify the design of a fault tolerant adder recently developed. Without these packages these designs could only be verified under fault operation by physically modifying the design to have a stuck-at fault. This would only verify the design for that one specific fault such that the methodology of manually injecting the faults does a poor job of verifying the design. The use of fbit or fsim_logic allows for the more rigorous evaluation of the design. As the designs indicate when an internal error is present but corrected it is possible to evaluate the ability to inject errors into the design even though the data output is correct. This is correct operation of the design and the aspect that requires these packages.

The design evaluated was a Single Error Correction/Dual Error Detection adder (SEC/DED). This design is capable of correcting any single error (on a bit-by-bit basis) and detecting dual errors (on a bit-by-bit basis). As the design uses a duplicated dual-rail logic implementation the error detection/correction takes place on each bit position so there could be multiple errors across a multiple but logic unit. The unit is very tolerant of errors. The design of a 2-bit SEC/DED is shown in Figure 7. There is significant circuit overhead but any SEC/DED architecture has this. The Five-way-redundant architecture for the computer systems of the space shuttle actually has more circuit overhead and provides slightly less protection that this methodology []. The advantage of this methodology is that the protection is ingained into the design at a very fine level of granularity, whereas the five-way-redundant methodology is implemented at a much high level of granularity. [10,11,12]

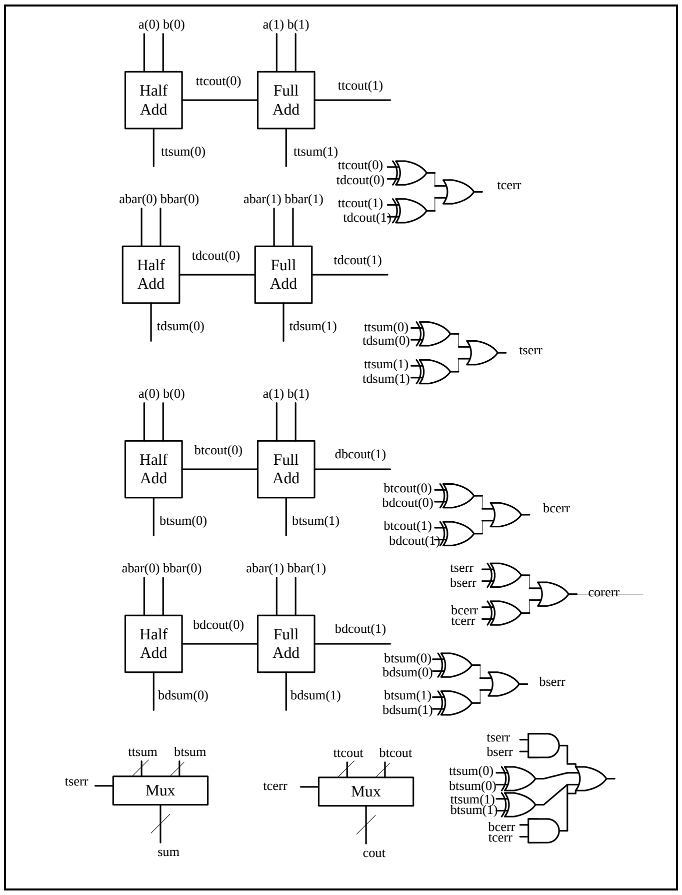
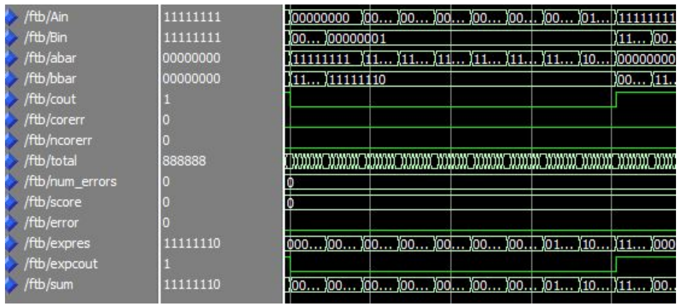
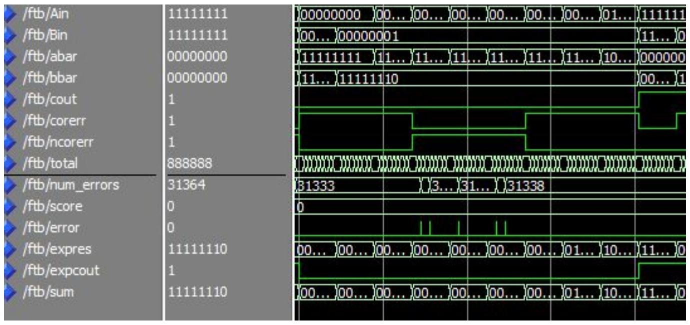


Figure 7. 2-bit SEC/DED adder

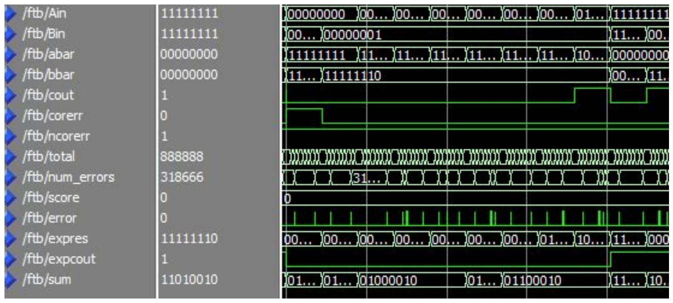
Various SEC/DED adder architectures were implemented up to an 8-bit version. All the adder were synthesized and synthesized cleanly. Figure 8, with multiple simulation waveforms, shows the waveform from the simulation of an 8-bit SEC/DED adder [13]. The waveforms illustrate the change in the number of errors injected as the threshold is adjusted from no error injection to a threshold of 0.70 where 30% of the gate level evaluations result in error. These waveform were extracted from a student report from my fall HDL Design and Verification class. The final project was to work on verification of the fault simulation packages fbit and fsim_logic and the use that package to verify the fault tolerant adder design.


```
/ftb/Ain          11111111
/ftb/Bin          11111111
/ftb/abar         00000000
/ftb/bbar         00000000
/ftb/cout         1
/ftb/corerr       0
/ftb/ncorerr      0
/ftb/total        888888
/ftb/num_errors   0
/ftb/score        0
/ftb/error        0
/ftb/expres       11111110
/ftb/expcout      1
/ftb/sum          11111110
```

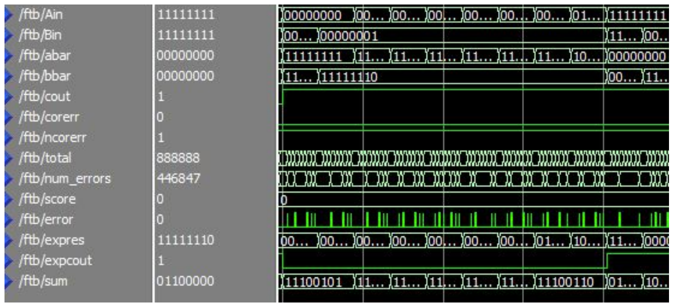
a) Simulation with no error



b) Simulation with tolerance = 0.99



b) Simulation with tolerance = 0.90



b) Simulation with tolerance = 0.70

Figure 8. Simulartion of 8-bit SEC/DED adder [13]

The students were provided with the code for the 8-bit SEC/DED adder design. The original VHDL had been written using TYPE BIT and all agreed that the modification need to use the fault simulation package was minimal.

V. Conclusions

The packages fbit_logic and fsim_logic have been presented. The need for such packages was presented. The packages have been used for the verification of fault tolerant digital circuit design and have shown these packages to be very effective for the verification of such designs. The packages can also be of significant benefit in the evaluation of the behavior of the performance of conventional designs under fault and aid in designing circuits that are more tolerant of intermittent faults.

Acknowledgment

I would like to acknowledge the student of my HDL Design and Verification class and their verification reports on the project to verify fibt, fsim_logic and the adder design.

References

[1] Richard D Schlichting and Fred B. Schneider, "Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems," ACM Transactions on Computing Systems, Vol 1, No. 3, August 1983.

[2] J. Renau, et. al., "SESC Simulator," January 2005. (http://sesc.sourceforge.net).

[3] Tim Miller, Radu Teordorescu, Naga Surapaneni, and Joanne DeGroat, "Flexible Redundancy in Robust Processor Architecture," Third Annual Graduate Student Poster Exhibition, Department of Computer Science and Engineering, The Ohio State University, Columbus, OH, April 9, 2009.

[4] Tim Miller, Naga Surapanneni, Radu Theodorescu, Joanne DeGroat, "Flexible Redundancy in Robust Processor Architecture," Weed 2009, Austin, Tx, June 20, 2009.

[5] US Patent 5896401, Miron Abramovici and Premachandran Rama Menon, "Fault simulator for digital circuitry," April 20, 1999.

[6] P.C. Ward and J.R. Armstrong, "Behavioral Fault Simulation in VHDL," 27th ACM/IEEE Design Automation Conference, Orlando, FL, 1990.

[7] Maria Isabel Ribeiro, "Gaussian Probability Density Functions: Properties and Error Characterization," http://users.isr.ist.utl.pt/~mir/pub/probability.pdf, 2004.

[8] Thara Rejimon and Sanjukta Bhanja, "An Accurate Probabilistic Model for Error Detection," 18th International Conference on VLSI Design, Kolkata, India, 2005.

[9] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008.

[10] Shailesh Niranjan and James F. Frenzel, "A Comparison of Fault-Tolerant State Machine Architectures for Space Born Electronics," IEEE Transactions on Reliability, Vol 45, No 1, March 1996.

[11] Gary Burke and Stephanie Taft, "Fault Tolerant State Machines", Jet Propulsion Laboratory, California Institute of Technology, Report D160/MALPD 2004.

[12] Keith S. Morgan, Daniel L McMurtrey, Brain H. Pratt and Michael J Wirthlin, "A Comparison of TMR With Alternative Fault-Tolerant Design Techniques for FPGAs," IEEE Transactions on Nuclear Science, Vol. 54, No. 6, December 2007.

[13] A, Antone, J Coles, M. Furst, S. Johnson, Y. Kulshrestha, "8-bit SEC/DED Adder and Fault Injection," Verification Report for ECE5462, Dept of ECE, Ohio State University, Dec 2012.